Assignment → 03

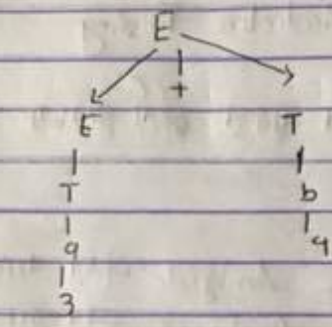
Main কাজ ২টা

১. Parse Tree Generate করা

২. Parse Tree থেকে Value Calculate করা।

Input যদি হয় 3+4

Parse Tree হবে



Grammar হবে

$E \rightarrow E+T$

$E \rightarrow T$

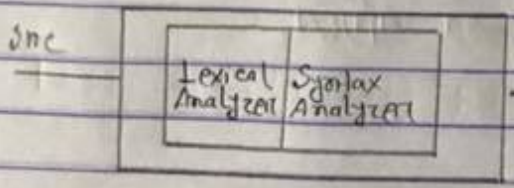
$T \rightarrow a$

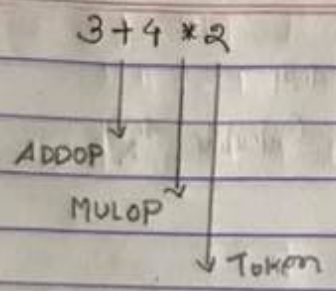
$T \rightarrow b$

→ Expression Related Grammar এর জন্য আমরা যেসব Operator use করবো, → +, −, *, /, (,)

→ Leaf node এর result Produce হবে।

Lexical Analyzer ⟶ Syntax Analyzer





Tool হিসেবে ব্যবহার করছি Predictive Parsing.

⇒ Token হিসেবে .l ফাইলে যাদের তৈরী করবো তাদেরকে return করবো।

⇒ .l ফাইল lexical Analyzer এর কাজ করবে,
.y ফাইল Syntax Analyzer এর কাজ করবে,

⇒ atoi মানে হল Array to integer integer.
yylval ভ্যারিয়েবল = atoi (yytext);

yylval এবং yytext হল global variable.
yylval token এর value ধরে রাখে,

⇒ দুইভাবে Token return করানো যায়,
a. এর আগে Symbol Pass
b. yylval এর মাধ্যমে,

⇒ return digit; মানে digit ক্যারাক্টারকে token হিসেবে Pass করবে।

⇒ yywrap ⇒ lex end of file এ গেছে কিনা সেটা জানার জন্য Call করা হয়।

⇒ yywrap যদি 0 return করে, তার মানে হল আরও একটা input file পড়তে হবে।

⇒ yywrap যদি 1 return করে, তার মানে, আর কোন input file নেই।

#define yywrap() 1 ⇒ internal error throw করবে।

⇒ extern লিখা মানে function এর body অন্য ফাইলে লিখা আছে।

⇒ token গুলোকে .y ফাইলে ডিক্লার দিতে হবে।

⇒ .y ফাইলে token গুলো specially operator গুলো Preceedence wise লিখা হয়, যে নিচে তার Precedence সবচেয়ে বেশি।

⇒ yyparse কে main থেকে call করা হয়। yyparse → yylex call করতে থাকে যতক্ষণ

a. কোন syntax error দেখা দেয়, তিবে yyerror call হয়।

b. 0 return করে যদি input process হয়ে যায়।

c. 1 return করে যদি syntax Error দেখা দেয়।

⇒ যে কোন Production এর Parent কে $ দিয়ে express করা হয়,

$ = $1 + $3

```
 a → .c
.g ——→ .e
   └→ y.h
```

⇒ Flex Bison উল্টোভাবে Parse Tree দেখায়,

⇒ yy debug   Parser এর trace রাখে,

```
Start: expn { Printf ("Found Expression, value = %d \n", $1); }
expn: expn '+' expn { Printf ("expn → expn (%d) + expn (%d) \n", $1, $3);
                      $$ = $1 + $3; }
| digit { Printf ("expn → digit (%d) \n", $1); }
```

Input →
3+3

expn → digit (3)
expn → digit (3)
expn → expn (3) + expn (3)